How to modify Golang's default FileServer to hide dot files

By
Marcus Willock

Twitter & Github handle: crazcalm

Blog: https://crazcalm.github.io/blog/

How to modify Golang's default FileServer to hide dot files

By
Marcus Willock

Twitter & Github handle: crazcalm

Blog: https://crazcalm.github.io/blog/

The Base Knowledge

- File server (in this context)
  - a server that allows other machines on the network access to the contents of its file system. Typically, you serve a directory, which give others access to everything in that directory (including nested directories)

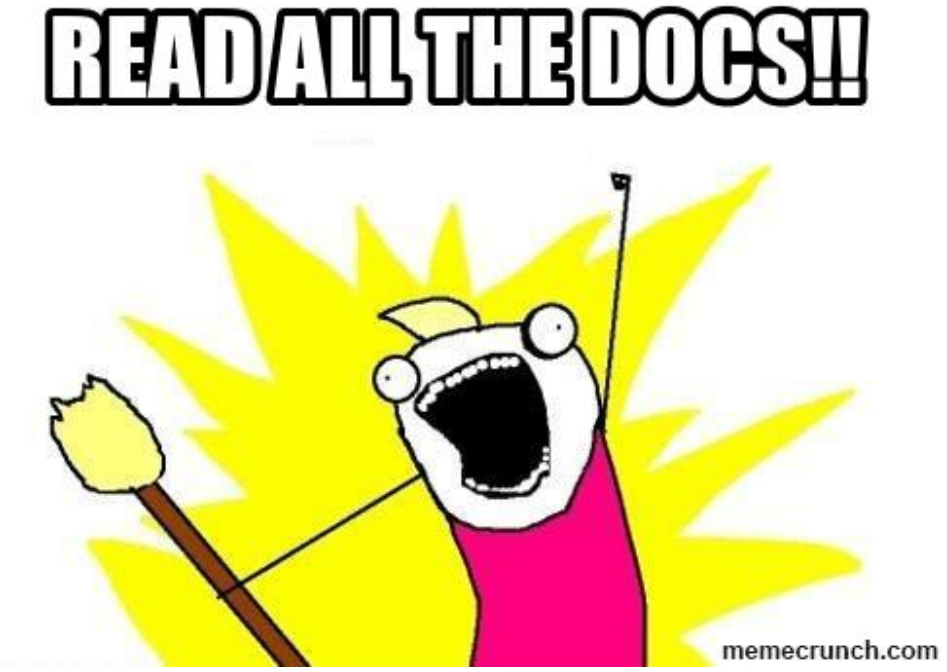
```
package main

import (
    "log"
    "net/http"
)

func main() {
    http.Handle("/", http.FileServer(http.Dir("/home/crazcalm")))
    log.Fatal(http.ListenAndServe(":12346", nil))
}
```

So many dot files...

First Thoughts

- All those dot files are annoying…
- I should be able to filter them out.
- There is probably a flag somewhere I can set to get rid of them.

Second Thought



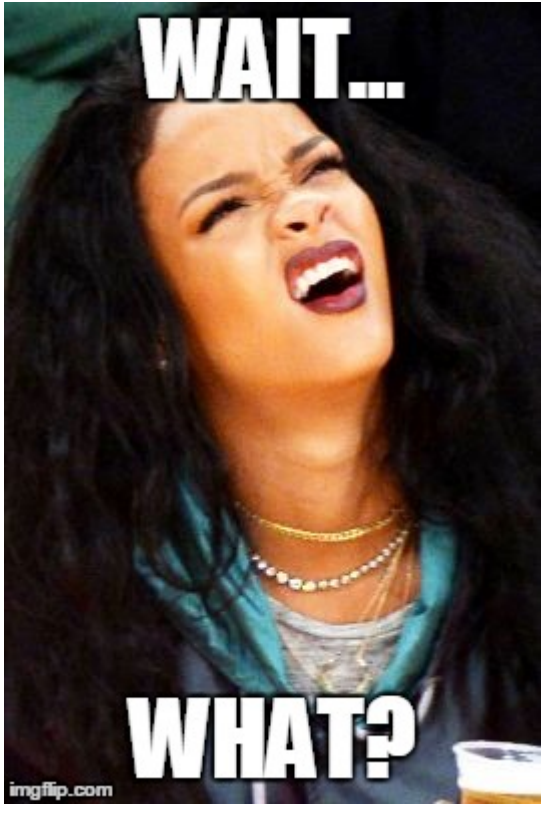
Go doc http.FileServer

```
package http // import "net/http"

func FileServer(root FileSystem) Handler
    FileServer returns a handler that serves HTTP requests with the contents of
    the file system rooted at root.

    To use the operating system's file system implementation, use http.Dir:

    http.Handle("/", http.FileServer(http.Dir("/tmp")))

    As a special case, the returned file server redirects any request ending in
    "/index.html" to the same path, without the final "index.html".
```

Go doc http.Dir

```
package http // import "net/http"

type Dir string
    A Dir implements FileSystem using the native file system restricted to a
    specific directory tree.

    While the FileSystem.Open method takes '/'-separated paths, a Dir's string
    value is a filename on the native file system, not a URL, so it is separated
    by filepath.Separator, which isn't necessarily '/'.

    Note that Dir will allow access to files and directories starting with a
    period, which could expose sensitive directories like a .git directory or
    sensitive files like .htpasswd. To exclude files with a leading period,
    remove the files/directories from the server or create a custom FileSystem
    implementation.

    An empty Dir is treated as ".".


func (d Dir) Open(name string) (File, error)
```

“create a custom FileSystem implementation...”



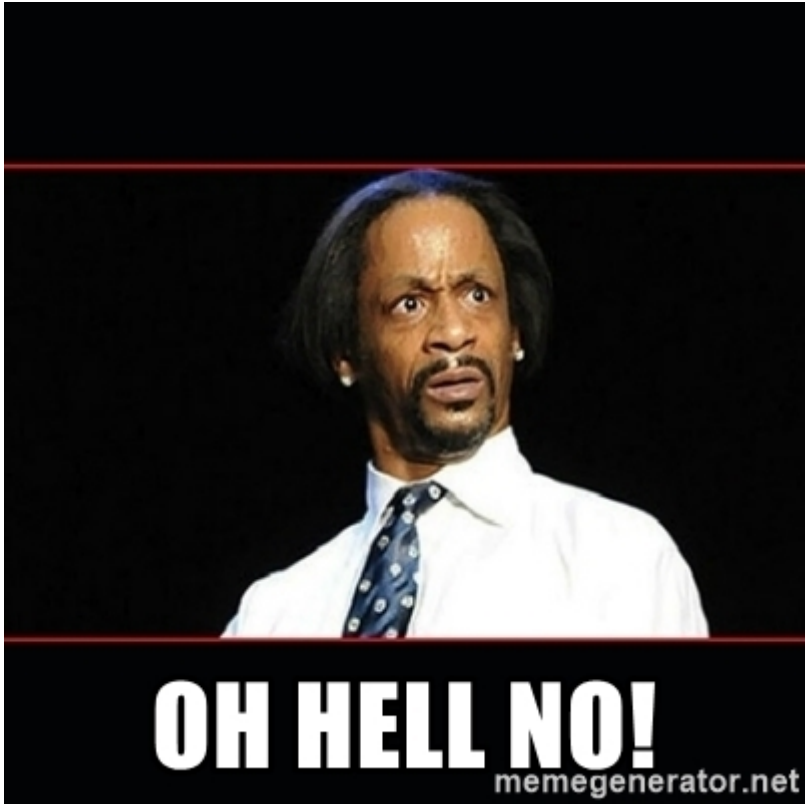
Go doc http.FileSystem

```
package http // import "net/http"

type FileSystem interface {
        Open(name string) (File, error)
}
    A FileSystem implements access to a collection of named files. The elements
    in a file path are separated by slash ('/', U+002F) characters, regardless
    of host operating system convention.
```

Implementing that doesn’t seem too bad, but what do they mean by “File”?

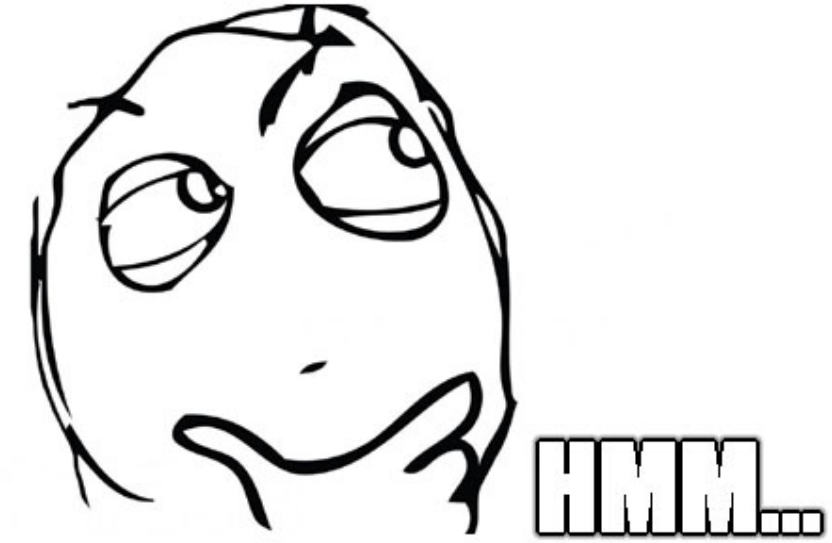
Go doc http.File

```
package http // import "net/http"

type File interface {
        io.Closer
        io.Reader
        io.Seeker
        Readdir(count int) ([]os.FileInfo, error)
        Stat() (os.FileInfo, error)
}
    A File is returned by a FileSystem's Open method and can be served by the
    FileServer implementation.

    The methods should behave the same as those on an *os.File.
```


OH HELL NO!
memegenerator.net

Plan B: Make a plan B

- There has to be an easier way.
- If I read the source code, I am bound to find something that will help me!



net/http/fs.go – FileServer

```
func FileServer(root FileSystem) Handler {
        return &fileHandler{root}
}

func (f *fileHandler) ServeHTTP(w ResponseWriter, r *Request) {
        upath := r.URL.Path
        if !strings.HasPrefix(upath, "/") {
                upath = "/" + upath
                r.URL.Path = upath
        }
        serveFile(w, r, f.root, path.Clean(upath), true)
}
```

net/http/fs.go – serveFile

```
func serveFile(w ResponseWriter, r *Request, fs FileSystem, name string, redirect bool) {
	const indexPage = "/index.html"

	// redirect .../index.html to .../
	// can't use Redirect() because that would make the path absolute,
	// which would be a problem running under StripPrefix
	if strings.HasSuffix(r.URL.Path, indexPage) {
		localRedirect(w, r, "./")
		return
	}

	f, err := fs.Open(name)
	if err != nil {
		msg, code := toHTTPError(err)
		Error(w, msg, code)
		return
	}
	defer f.Close()

	d, err := f.Stat()
	if err != nil {
		msg, code := toHTTPError(err)
		Error(w, msg, code)
		return
	}

	if redirect {
		// redirect to canonical path: / at end of directory url
		// r.URL.Path always begins with /
		url := r.URL.Path
		if d.IsDir() {
			if url[len(url)-1] != '/' {
				localRedirect(w, r, path.Base(url)+"/")
				return
			}
		} else {
			if url[len(url)-1] == '/' {
				localRedirect(w, r, "../"+path.Base(url))
				return
			}
		}
	}

	// redirect if the directory name doesn't end in a slash
	if d.IsDir() {
		url := r.URL.Path
		if url[len(url)-1] != '/' {
			localRedirect(w, r, path.Base(url)+"/")
			return
		}
	}

	// use contents of index.html for directory, if present
	if d.IsDir() {
		index := strings.TrimSuffix(name, "/") + indexPage
		ff, err := fs.Open(index)
		if err == nil {
			defer ff.Close()
			dd, err := ff.Stat()
			if err == nil {
				name = index
				d = dd
				f = ff
			}
		}
	}

	// Still a directory? (we didn't find an index.html file)
	if d.IsDir() {
		if checkIfModifiedSince(r, d.ModTime()) == condFalse {
			writeNotModified(w)
			return
		}
		w.Header().Set("Last-Modified", d.ModTime().UTC().Format(TimeFormat))
		dirList(w, r, f)
		return
	}

	// serveContent will check modification time
	sizeFunc := func() (int64, error) { return d.Size(), nil }
	serveContent(w, r, d.Name(), d.ModTime(), sizeFunc, f)
}
```

- This function performs a number of checks around whether or not it is dealing with a “valid” directory.
- The last if statement deals with our case.

ServeFile func snippet

```
        // Still a directory? (we didn't find an index.html file)
        if d.IsDir() {
                if checkIfModifiedSince(r, d.ModTime()) == condFalse {
                        writeNotModified(w)
                        return
                }
                w.Header().Set("Last-Modified", d.ModTime().UTC().Format(TimeFormat))
                dirList(w, r, f)
                return
        }

        // serveContent will check modification time
        sizeFunc := func() (int64, error) { return d.Size(), nil }
        serveContent(w, r, d.Name(), d.ModTime(), sizeFunc, f)
}
```

dirList looks like HOPE!!!!

DirList – Source

```
func dirList(w ResponseWriter, r *Request, f File) {
        dirs, err := f.Readdir(-1)
        if err != nil {
                logf(r, "http: error reading directory: %v", err)
                Error(w, "Error reading directory", StatusInternalServerError)
                return
        }
        sort.Slice(dirs, func(i, j int) bool { return dirs[i].Name() < dirs[j].Name() })

        w.Header().Set("Content-Type", "text/html; charset=utf-8")
        fmt.Fprintf(w, "<pre>\n")
        for _, d := range dirs {
                name := d.Name()
                if d.IsDir() {
                        name += "/"
                }
                // name may contain '?' or '#', which must be escaped to remain
                // part of the URL path, and not indicate the start of a query
                // string or fragment.
                url := url.URL{Path: name}
                fmt.Fprintf(w, "<a href=\"%s\">%s</a>\n", url.String(), htmlReplacer.Replace(name))
        }
        fmt.Fprintf(w, "</pre>\n")
}
```

Options!

- Change the output of f.Readdir(-1) so that it does not include dot files
- Modify the loop so none of the dot files are written to the response!

OR

Option 2 please!

```
func dirList(w http.ResponseWriter, r *http.Request, f http.File) {
        dirs, err := f.Readdir(-1)
        if err != nil {
                logf(r, "http: error reading directory: %v", err)
                http.Error(w, "Error reading directory", http.StatusInternalServerError)
                return
        }
        sort.Slice(dirs, func(i, j int) bool { return dirs[i].Name() < dirs[j].Name() })

        w.Header().Set("Content-Type", "text/html; charset=utf-8")
        fmt.Fprintf(w, "<pre>\n")
        for _, d := range dirs {
                name := d.Name()

                //Added by Marcus
                if !*showDotFiles && strings.HasPrefix(name, ".") {
                        continue
                }

                if d.IsDir() {
                        name += "/"
                }
                // name may contain '?' or '#', which must be escaped to remain
                // part of the URL path, and not indicate the start of a query
                // string or fragment.
                url := url.URL{Path: name}
                fmt.Fprintf(w, "<a href=\"%s\">%s</a>\n", url.String(), htmlReplacer.Replace(name))
        }
        fmt.Fprintf(w, "</pre>\n")
}
```

(Theoretically) Done

- I find it taboo to modify the standard library, so I copied all the needed code for FileServer and put it in a file.

- I copied more than 1,000 lines of code from the net/http package… (serve-it.go source)

Closing thoughts

- Mission accomplished!
- But why do I still feel like I am missing something...
- **Thinks out loud**
- “Just because you cannot see something, does not mean you cannot access it...”

Need to kill the request for dot files!

```
func (f *fileHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
        upath := r.URL.Path
        fmt.Printf("Requested path: %s\n", upath)
        if !strings.HasPrefix(upath, "/") {
                upath = "/" + upath
                r.URL.Path = upath
        }

        //Added by Marcus
        if !*showDotFiles {
                pathParts := strings.Split(r.URL.Path, "/")
                for _, part := range pathParts {
                        if strings.HasPrefix(part, ".") {
                                http.Error(w, "403 Forbidden", http.StatusForbidden)
                                return
                        }
                }
        }

        serveFile(w, r, f.root, path.Clean(upath), true)
}
```


Mission
Accomplished!
Meme creator.org